MASTER EN SCIENCES MATHÉMATIQUES

Reconnaissance dynamique de formules mathématiques

ANCIAUX, Jérôme

Award date:
2002

Awarding institution:
Universite de Namur

FUNDP
Faculté des Sciences
Département de Mathématique

Rempart de la Vierge, 8
B-5000 Namur Belgique

Reconnaissance dynamique de formules mathématiques

Mémoire présenté pour l'obtention
du grade de
Licencié en Sciences
Mathématiques
par

Jérôme ANCIAUX

Promoteur : M. REMON s.j.

Année académique 2001-2002

Reconnaissance dynamique de formules mathématiques

Résumé:

L'application des techniques de reconnaissance de caractères aux formules mathématiques nous oblige à modifier le rôle des bases d'entrainement habituellement utilisées dans la reconnaissance de texte. En effet dans le cas de formules mathématiques, cette base d'entrainement devrait être quasiment infinie. Dans ce travail, nous développons une méthode dynamique de reconnaisance des formules mathématiques. Basée sur un découpage par projections et une distance entre deux images équivalente à la proportion de pixels communs, la méthode que nous utilisons propose des alternatives aux entités reconnues. Ces alternatives permettent de traiter, entre autre, les cas des exposants, des indices et des mauvaises reconnaissances. L'implémentation de ces méthodes donne de bons résultats, même dans des cas de formules complexes.

Dynamic recognition of mathematical formulas

Abstract:

The application of the techniques of character recognition to mathematical formulas requires us to change the role of training sets usually used in text recognition. Indeed, in the case of mathematical formulas those training sets should be almost infinite. In this work, we develop a dynamic method for recognition of mathematical formulas. Based on a cutting out by projections and a distance between two pictures equivalent to the proportion of common pixels, the method we use proposes alternatives to recognized entities. Those alternatives allow to deal with, among others, cases of exponents,suffixes, and erroneous recognitions. The implementation of those methods gives good results, even in cases of complex formulas.

Je tiens à remercier les personnes sans lesquelles ce mémoire n'aurait pas été possible:
Monsieur Rémon, promoteur de ce mémoire, pour ses indications et sa confiance.
Fabian Bastin pour ses recherches sur le format TIFF et la librairie LIBTIFF.
Jean-Yves Pirçon pour ses indications sur les accès fichier en C++.
Vincent Bertholet pour son debugage de dernière minute.
Boule, alias pcdid-4 alias 138.48.148.108, pour les heures de travail acharné

Je tiens aussi à remercier les personnes sans lesquelles *rien* n'aurait été possible:
Loulou, Caillou, Lokum et Jacquouille pour leur patience.
Mili pour tout et pour le reste.

Tous ceux et celles qui, de près où de loin, ont fait que se soit un réel plaisir:
Louise, Marie et Catherine, l'Echomatheux, Flying Nigga, la troupe du ski et bien d'autres...

Introduction

Avant le développement des documents numériques, la diffusion et l'archivage des informations à tout sujet étaient effectués sur papier. Mais depuis l'essor de la numérisation des documents, il devient impératif de pouvoir récupérer l'information toujours stockée sur papier et de l'intégrer aux documents numériques existants. Pas seulement en vue de limiter l'usage de document papier car son usage reste préferé lorsqu'il s'agit de consulter une grande quantité d'information ou plus simplement de laisser preuve d'un contrat quelconque. Mais surtout pour assurer la portabilité des informations sur papier vers les ordinateurs sans devoir recommencer toute l'étape fastidieuse, car manuelle, de l'encodage de l'information. L'étape ultime du processus étant qu'un ordinateur soit capable de traiter avec autant d'efficacité un document sur papier qu'un document numérique.

L'analyse et la reconnaissance de documents s'attachent à l'identification de textes, de graphiques ou de toutes autres notations composant une image ainsi qu'à l'extraction d'information comme le ferait un lecteur humain. L'analyse de documents se divise en deux grands ensembles:

- l'analyse du texte
- l'analyse des composants graphiques

La partie qui s'emploie à reconnaître les caractères ou les structures du texte (colonnes, paragraphes,...) et à déterminer d'éventuels angles de rotation s'appelle *l'analyse textuelle.* Pour tout ce qui concerne la reconnaissance de

lignes, de schémas, de symboles constituant un diagramme, de l'identification de logos d'entreprise, etc, il s'agit de *l'analyse graphique.*

A l'heure actuelle, il est déjà possible de convertir un document papier en document électronique au moyen de systèmes et algorithmes développés dans le cadre de l'analyse et de la reconnaissance de documents. L'utilisation de ces systèmes couplés à des lecteurs optiques est déjà fort répandue:

- en bureautique, où la reconnaissance de textes imprimés est devenue courante grâce à l'essor des OCR (optical character recognition).
- dans le milieu bancaire, où l'on utilise des lecteurs de chèques.
- dans les services postaux, avec l'automatisation du tri du courier par lecture du code postal.

On distingue actuellement deux grandes classes de systèmes de traitement de documents:

- les applications spécifiques: lecture de plans, interprétations de circuits électriques ou de vues aériennes,...
- les systèmes plus généraux

Les premières applications sont conçues de telle sorte qu'elles prennent en compte le type de document à analyser, tandis que les secondes sont plus générales et ont pour but de traiter un grand nombre de documents. Même si pour le texte traditionnel ces logiciels sont assez complets et leurs résultats plutôt satisfaisants, certains types ou composants de documents posent problème et nécessitent l'emploi de sous-systèmes plus spécifiques.

Un exemple de ces types problématiques est celui employant les notations mathématiques. Par leur diffusion importante dans la plupart des documents scientifiques, ces formules mathématiques renferment une part importante d'information et de connaissance qui est dès lors perdue pour le lecteur. Une reconnaissance automatique (sans saisie manuelle) des formules mathématiques permettrait aussi une exportation plus facile vers les systèmes de

calcul ou de représentation graphique.

Difficultés par rapport à la reconnaissance de textes

Beaucoup de recherches ont déjà été effectuées dans le domaine de la reconnaissance optique de caractères (OCR optical character recognition). Certains logiciels d'OCR sont déjà commercialisés avec des résultats plus que satisfaisants. Mais ces produits sont décevants lorsqu'on les emploie pour la reconnaissance de formules. Par exemple, si nous soumettons à un programme de reconnaissance les formules suivantes générées en LaTeX:

$$\frac{sin(x)}{cos(x)}$$

$$\int_{-\infty}^{+\infty} e^x dx$$

$$\sum_{x=1}^{+\infty} \frac{3x}{\int_{-\infty}^{+\infty} e^x dx}$$

on obtient le résultat de la figure 1:

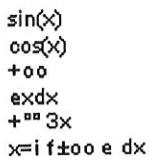

FIG. 1 – *Résultat du programme de reconnaissance.*

Le résultat n'est évidemment pas très convaincant. En plus de caractères spéciaux non reconnus comme $\sum$ ou $\int$, les exponentiations, les indexations et les fractions ne sont pas reconnues. La différence essentielle entre le texte traditionnel et les formules mathématiques consite dans le fait qu'un texte traditionnel ne souffre pas d'un agencement particulier des caractères qui

le composent. Les caractères d'un texte se suivent avec des blancs et des sauts de lignes, tandis que pour une formule mathématique, les ensembles de caractères se superposent (fractions, indices pour une somme,...), peuvent prendre plus d'une ligne (intégrale, parenthèses de matrice,...), se mettre en exposant ou en indice. On parle d'un arrangement bidimensionnel pour les formules mathématiques. Ces dernières ont donc une structure particulière et nécessitent dès lors un traitement particulier.
En conclusion, pour être capable d'interprêter une formule mathématique, il faut traiter deux problèmes distincts: l'analyse des symboles qui composent la formule et l'analyse de l'agencement de ces symboles.

Etat des lieux

Certaines recherche ont déjà été effectuées dans le cadrez de reconnaissance formule mathématiques, notamment par Stéphane Lavirotte dans sa thèse intitulée *Reconnaissance structurelle de formules mathématiques typographiées et manuscrites* [9]. Un programme de reconnaissance a d'ailleurs été développé dans le cadre de cette thèse. La reconnaissance utilisée dans cette thèse est basée sur les grammaires formelles.

Optique de notre travail

Le but de ce mémoire est le développement d'une méthode dynamique de reconnaissance de formules mathématiques. La reconnaissance ne serait plus basée sur des grammaires formelles mais proposerait une reconnaissance de base en proposant des solutions alternatives.

Plan du travail

Le travail est découpé en sept partie:

1. Les images TIFF et la librairie LIBTIFF
2. Découpage
3. Distance et alphabet
4. Regroupement des entités
5. Applications
6. Conclusion et pistes de réflexion

Les images TIFF et la librairie LIBTIFF

Nous y expliquerons les choix que nous avons faits en matière de langage de programmation, de format d'images traitées et de librairie utilisée pour traiter ce format.

Découpage

Nous y expliquerons plus en détails le découpage que nous avons développé. Ce découpage est proche du découpage par projections proposé par M. Okamoto ([13], [14] et [15]). Nous détaillerons aussi l'étape de reconnaissance: une entité sera associée à l'image de l'alphabet qui lui sera la plus proche.

Distance et alphabet

Nous y expliquerons la distance utilisée lors de l'étape de reconnaissance ainsi que les critères retenus pour l'élaboration de notre alphabet. La distance entre deux images vaut la proportion de pixels communs entre ces deux images.

Regroupements des entités

Nous y expliquerons comment nous recréons les expressions et les alternatives sur base de l'arbre créé dans l'étape de découpage.

Applications

Nous présenterons les résultats obtenus par notre programme sur des exemples de différentes origines:

1. LaTeX
2. Word
3. Manuscrite

Chapitre 1

Les images TIFF et la librairie LIBTIFF

1.1 Choix du format d'image

Dans un premier temps nous avions commencé à travailler avec des images BMP en utilisant des sous-routines de lecture et d'écriture pour le langage *fortran77*. Mais ces sous-routines étaient trop exigeantes en ce qui concerne le format sous lequel les formules devaient être sauvées. Après avoir cherché, il s'est avéré que le format TIFF était plus facile à traiter et que les langages *C* et *C++* offraient l'avantage d'une abondance de librairies de traitement des images de format TIFF.

1.2 LIBTIFF

LIBTIFF est une librairie graphique pour le language *C* qui fournit les supports pour traiter le "Tag Image File Format" (TIFF). La librairie fournit le moyen d'accéder et de créer facilement des images TIFF. Cette librairie et ses tutoriaux sont disponibles sur internet ([1] et [2]). Voici une présentation rapide des fonctions de la librairie qui ont été utilisées pour l'élaboration de

notre programme. Nous distinguons deux types dans ces fonctions:

- Les fonctions d'ouverture pour la lecture d'images TIFF
- Les fonctions d'ouverture pour la création d'images TIFF

1.2.1 Fonctions d'ouverture pour la lecture d'images TIFF

Il faut tout d'abord spécifier le nom de l'image que nous allons traiter:

```
TIFF *tif=TIFFOpen(''nomdelimage.tif'',''r'');
```

Le `r` spécifie que l'image est ouverte en lecture (read).
Il faut ensuite lire la taille de l'image. Autrement dit le nombre de pixels sur la largeur (le nombre de lignes de l'image) et sur la longueur (le nombre de colonnes de l'image):

```
TIFFGetField(tif,TIFFTAG_IMAGEWIDTH,&nbrligne);
TIFFGetField(tif,TIFFTAG_IMAGELENGTH,&nbrcolonne);
```

Nbrligne et nbrcolonne sont de types entiers longs non signés. L'étape suivante permet d'allouer la place mémoire pour stocker l'image:

```
unsigned long npixels=nbrligne*nbrcolonne;
raster=(uint32*)_TIFFmalloc(npixels*sizeof(uint32));
```

C'est le raster qui contient toute l'information concernant les pixels de l'image. Maintenant que nous avons alloué la place nécessaire pour stocker le raster il faut le lire:

```
TIFFReadRGBAImage(tif,nbrligne,nbrcolonne,raster,0);
```

Nous devons toujours vérifier la valeur de retour de la fonction pour nous assurer qu'aucune erreur n'a été rencontrée durant la procédure de lecture. Une valeur de retour 1 signifie que la lecture s'est déroulée sans problème, la valeur 0, signifie l'existence d'une erreur lors de la procédure. Nous disposons maintenant d'un tableau (raster) contenant les valeurs des

pixels. Un élément du tableau correspond à un pixel de l'image, et chaque pixel possède une valeur de 4-bytes. Chacun des bytes représente un des canaux de la valeur du pixel (Red Green Blue Alpha). Chaque canal est représenté par une valeur entière comprise entre 0 et 255. Pour accéder aux canaux de manière individuelle, nous utilisons la fonction:

```
char red=(char)TIFFGetR(raster[i]); ou
int red=(char)TIFFGetR(raster[i]);
```

Le R peut être remplacé par un G, un B, ou un A suivant que nous voulions accéder au canal vert, bleu, ou alpha. i est l'indice du pixel dans le raster.

Remarque: Il est important de remarquer que le raster est un tableau unidimensionnel tandis que nous représentons des images bi-dimensionnelles. Le raster stocke les lignes les unes à la suite des autres en commençant par la ligne du bas.

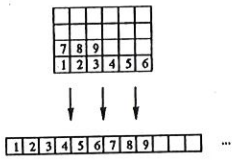

FIG. 1.1 – *Correspondance entre une image bidimensionnelle et le raster.*

Lorsque nous disposons de toutes les informations nécessaires, nous devons détruire le raster et libérer la mémoire au moyen de la fonction:

```
_TIFFfree(raster);
```

1.2.2 Fonction d'ouverture pour la création d'images TIFF

Il faut également commencer par spécifier le nom de l'image:

```
TIFF *out=TIFFOpen(''nomdelimage.tif'',''w'')
```

Le `w` signifie que l'image est ouverte en écriture (write). Si le nom spécifié renvoit à une image existante, celle-ci sera remplacée, sinon une autre image sera créée. Nous devons stocker les informations sur les canaux des pixels que nous utiliserons dans un tableau de caractères. Il faut une case pour chaque canal de chaque pixel[1]:

```
int sampleperpixel=4;
char *image=new char[nbrligne*nbrecolonne*sampleperpixel];
```

Si nous désirons que nos images n'aient pas de canal Alpha, la valeur pour sampleperpixel sera fixée à 3. Nous devons ensuite fixer les paramètres de l'image:

```
TIFFSetField(out,TIFFTAG_IMAGEWIDTH,nbrligne);
TIFFSetField(out,TIFFTAG_IMAGELENGTH,nbrcolonne);
```

fixent le nombre de ligne et le nombre de colonne de l'image.

```
TIFFSetField(out,TIFFTAG_SAMPLERPIXEL,sampleperpixel);
```

fixe le nombre de canaux par pixel.

```
TIFFSetField(out,TIFFTAG_BITSPERSAMPLE,8);
```

fixe la taille des canaux.

```
TIFFSetField(out,TIFFTAG_ORIENTATION,ORIENTATIONTOPLEFT);
```

fixe l'orientation de l'image. Nous utilisons d'autres fonctions pour configurer l'image:

```
TIFFSetField(out,TIFFTAG_PLANARCONFIG,PLANARCONFIG_CONTIG);
TIFFSetField(out,TIFFTAG_PHOTOMETRIC,PHOTOMETRIC_RGB);
```

1. dans l'ordre RGBA

Une fois que le tableau image contient les informations sur les canaux de l'image que nous voulons créer, nous utilisons d'autres fonctions pour procéder au stockage des données. Le stockage s'effectue ligne par ligne. Nous devons d'abord connaître la longueur mémoire nécessaire pour une ligne:

```
tsize_t linebytes=sampleperpixel*nbrcolonne;
```

Nous utilisons ensuite un pointeur pour stocker la ligne que nous sommes en train de mettre dans l'image. Et nous allouons à ce pointeur la mémoire calculée précédemment:

```
unsigned char *buf=NULL;
if(TIFFScanlineSize(out)==linebytes)
buf=(unsigned char *)_TIFFmalloc(linebytes);
else
buf=(unsigned char *)_TIFFmalloc(TIFFScanlineSize(out));

TIFFSetField(out,TIFFTAG_ROWERSTRIP,TIFFDefaultStripSize(out,nbrcolonne*
sampleperpixel);
```

Nous écrivons l'image ligne par ligne en utilisant buf pour le stockage temporaire.

```
for (uint32 row=0; row<nbrligne;row++)
{
memcpy(buf,&image[(nbrligne-row-1)*linebytes],linebytes);
if (TIFFWriteScanline(out,buf,row,0)<0)
break
}
```

Enfin, nous détruisons le buffer et fermons l'image nouvellement créée. La librairie LIBTIFF comporte d'autres fonctions pour le traitement des images TIFF mais nous ne les détaillerons pas ici.

Chapitre 2

Traitement des images

2.1 L'acquisition des données

L'acquisition des données est l'étape durant laquelle nous allons transformer la formule que nous désirons traiter en format directement exploitable par nos programmes. En l'occurence, le format TIFF. De manière générale l'acquisition s'effectue par numérisation. L'étape de numérisation est donc une étape primordiale avant d'opérer une quelconque reconnaissance. La qualité de l'image dépend de celle de la nummérisation. Il existe certains problèmes fréquemment rencontrés lors de cette étape.

2.1.1 Seuil de numérisation

L'une des principales difficultés survenant lors de la numérisation d'un document est le choix du **seuil de numérisation** (thresholding). Le choix du contraste et de la luminosité peut influencer la qualité de l'image. Si le seuil est trop bas, nous remarquerons un manque de points dans les zones sombres. Dans le cas contraire, le nombre de points parasites dans l'image augmentera. Dans les deux cas, la difficulté de l'étape de reconnaisance sera accrue. Ce problème n'étant pas spécifique aux formules mathématiques, plusieurs méthodes ont déjà été développées afin de le résoudre (voir par

exemple [17], [16] et [11]).

2.1.2 Réduction du bruit

La numérisation peut donner cours à un bruitage de l'image, il faut donc procéder à un filtrage de l'image après celle-ci. Ce problème est très fréquent lorsque la numérisation s'effectue sur des documents de mauvaise qualité. Le bruit prend la forme de pixels noirs isolés ou regroupés en petites taches. Ce bruit introduisant une perturbation, il est primordial de traiter le problème. D'autant plus que dans le cas des formule mathématiques, certains symboles de petite taille peuvent être interpretés comme étant du bruit. Ce problème a également déjà fait l'objet de maintes recherches (voir par exemple [17], [12], [18] et [3]). Cependant, aucune technique n'a été spécialement développée pour le cas des formules mathématiques.

2.1.3 Réalignement de l'image

Si lors de la numérisation le document papier est mal positionné, l'image créée est "penchée". Il faudra donc réorienter l'image afin de ne pas perturber la reconnaissance. Les techniques qui ont été développées pour résoudre le problème dans le cas de texte traditionnel, sont performantes pour les formules mathématiques. Il est bon cependant de remarquer que dans le cas de formules isolées, ces techniques rencontrent certains problèmes dûs au manque de linéarité dans les formules (contrairement au texte traditionnel). Mais étant donné que le but final est de reconnaître des formules dans un contexte textuel, ces problèmes ne sont pas trop insurmontables.

2.1.4 Isolement d'une formule mathématique

Comme précisé juste avant, les expressions mathématiques sont généralement inclues dans un contexte textuel, soit en tant que formule mise en valeur, donc isolée du reste du texte, soit directement au fil du texte. Il faut

donc, après le traitement de l'image, procéder à l'identification et à la séparation de la formule du reste du document. Cependant pour plus d'informations voir [6], [10] et [7].

2.1.5 Acquisition des formules dans ce mémoire

Dans ce mémoire, nous n'effectuerons pas de numérisation puisque les formules que nous traiterons seront directement dans le format souhaité. En effet, elles sont générées par un logiciel LaTeX ou Word et directement capturées à l'écran. Nous ne recontrerons donc pas les problèmes dûs au passage du format papier au format numérique: le choix du seuil de numérisation, l'apparition de bruit, ou le mauvais alignement des formules. Nos formules étant générées individuellement, nous ne devrons pas non plus procéder à leur isolement du reste du texte. Toutefois, nous rencontrerons des problèmes propres à notre méthode d'acquisition des données. Ceux-ci et les solutions que nous leur avons apportées sont détaillés ci-après.

2.2 Centrage de l'image

Le premier de ces traitements consiste en un découpage de l'image capturée. En effet, la capture s'effectuant manuellement, la zone de blanc entourant l'image est variable. Cette zone ne comportant aucune information nécessaire à la reconnaissance, nous avons développé un programme pour supprimer cette zone de blanc. Sur base de l'image résultat de notre capture manuelle, le programme crée une image dont le contour blanc a été enlevé[1].

2.2.1 Stratégie

La stratégie utilisée est la suivante: nous parcourons l'image horizontalement du haut vers le bas jusqu'au moment où nous trouvons le pixel noir

1. La ligne bleue représente la délimitation de l'image.

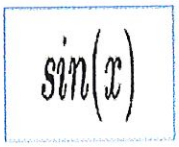

FIG. 2.1 – *Résultat de la capture.*

FIG. 2.2 – *Image après l'exécution du programme.*

le plus haut dont nous gardons le numéro de ligne. Nous faisons de même du bas vers le haut afin de trouver la ligne du pixel le plus bas. En parcourant l'image verticalement de gauche à droite (resp. de droite à gauche), nous trouvons la colonne du pixel noir le plus à gauche (resp. à droite). Nous créons ensuite une image en ne reprenant que les pixels à l'intérieur de la zone délimitée par ces quatres pixels.
En plus de créer une nouvelle image, le programme renvoit également quatres valeurs entières: `plushaut`, `plusbas`, `plusdroit` et `plusgauche`.

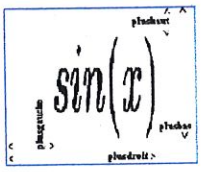

FIG. 2.3 – *Expliquation de* `plushaut`, `plusbas`, `plusdroit` *et* `plusgauche`.

2.2.2 Prototype de `captur.cpp`

```
void captur(char nomimage[256],char nomnewimage[256],int &plusbas
    ,int &plushaut,int &plusgauche,int &plusdroit);
```

- `nomimage` est une chaîne de caractères contenant le nom de l'image à traiter.
- `nomnewimage` est une chaîne de caractères contenant le nom de l'image après traitement.
- `plushaut` (resp. `plusbas`, `plusdroit` et `plusgauche`) est un entier, après l'exécution de la fonction, la valeur vaut le nombre de lignes entre le bord supérieur de l'image et le haut de la formule (resp. le nombre de lignes entre le bord supérieur de l'image et le bas de la formule, le nombre de colonnes entre le bord gauche de l'image et le nombre de colonnes entre le bord gauche de l'image et le bord gauche de la formule).

Problème rencontré

Nous avons rencontré un problème à cette étape du processus. En effet, même si LaTeX génère des formules en noir et blanc, le logiciel de visualisation les affiche avec des niveaux de gris. Nous avons donc dû imposer un seuil au delà duquel nous considérions le pixel comme non-blanc. Ce seuil a été fixé à 210. Nous considérons donc un pixel comme non-blanc si les valeurs de ses trois composantes excèdent 210.

2.2.3 Code de `captur.cpp`

Pour le code du programme se référer au volume d'annexes.

2.3 Coloriage en noir et blanc

Suite au problème rencontré lors de l'étape de centrage de l'image avec l'apparition de niveaux de gris par l'intermédiaire du logiciel de visualisation, nous avons développé un programme qui transforme l'image avec niveau de gris en une image noir et blanc de même taille.

2.3.1 Prototype de `noirblanc.cpp`

```
void noirblanc(char nomimage[256],char nomnewimage[256]);
```

- `nomimage` est une chaîne de caractères contenant le nom de l'image à traiter.
- `nomnewimage` est une chaîne de caractères contenant le nom de l'image après traitement.

2.4 Redimensionnement de l'image

Dans une étape ultérieure, nous serons amenés à comparer les images issues du découpage de notre formule avec des images de notre ensemble de référence. Notre tâche sera facilitée si les images à comparer sont de taille identique. De plus, pour la pertinence de la distance utilisée, il serait bon de pouvoir agrandir les images. A cette fin nous avons développé un programme qui, sur base d'une image, recrée une image ayant la longueur et la largeur désirées.

2.4.1 Stratégie

Soit a (resp. b), la longueur (resp. la largeur) de l'image à traiter, et soit A (resp. B), la longueur (resp. la largeur) désirée, le programme crée une image en faisant correspondre au pixel de coordonnées (i,j) de la nouvelle image, la couleur du pixel de coordonnées $(\lceil(\frac{x}{A} * a)\rceil,\lceil(\frac{y}{B} * b)\rceil)$[2].

2. $\lceil\ \rceil$ équivaut à la partie entière inférieure

2.4.2 Prototype de `resize.cpp`

```
void resize(char nomimage[256],char nomnewimage[256],
    int longueur,int largeur);
```

- `nomimage` est une chaîne de caractères contenant le nom de l'image à traiter.
- `nomnewimage` est une chaîne de caractères contenant le nom de l'image après traitement.
- `longueur` (resp. `largeur`) est un entier contenant le nombre de colonnes (resp. lignes) de l'image après traitement

2.4.3 Code de `resize.cpp`

Pour le code du programme se référer au volume d'annexes.

Chapitre 3

Découpage

L'étape de découpage va nous permettre d'isoler chacun des symboles de la formule en gardant sa localisation. L'idée est que pour chaque symbole constituant la formule nous désirons l'inscrire dans un bloc en gardant les coordonnées de ce bloc dans l'image globale.

3.1 Création d'une matrice image

Nous avons déjà spécifié dans le chapitre expliquant la librairie LIBTIFF, que les sous-routines de lecture d'images TIFF nous renvoyaient un tableau dont la taille égale le nombre de pixels de l'image. Dans ce tableau les lignes sont juxtaposées en commençant par la ligne du bas de l'image. Ce type de représentation ne facilite pas l'accès à un pixel. En effet il serait plus facile d'avoir une représentation bidimensionnelle de notre image. Pour ce faire nous avons développé une sous-routine qui transforme le raster (tableau dont la taille vaut la longueur * la largeur de l'image) en une matrice ayant le même nombre de lignes et de colonnes que notre image. Comme nous ne traitons que des images noir et blanc, il n'est pas nécessaire de garder les quatres canaux[1]. La valeur d'un élément de la matrice vaut donc 1 si le pixel

1. RGBA

est noir et 0 sinon. Cette sous-routine utilise une sous-routine développée par F.Bastin pour l'allocation de la mémoire.

3.1.1 Prototype de `creationmatrice.cpp`

```
void creationmatrice(char nom[256],double ***mat,uint32& nlig
,uint32& ncol);
```

- `nom` est une chaîne de caractères contenant le nom de l'image à traiter.
- `***mat` est pointeur de type double, après l'exécution de la fonction il désigne la matrice créée. Pour accéder à l'élément ij de celle-ci nous utilisons `mat[i][j]`.
- `nlig` (resp. `ncol`) est un entier long non signé contenant le nombre de lignes (resp. colonnes) de la matrice.

3.1.2 Code de `creationmatrice.cpp`

Pour le code du programme se référer au volume d'annexes.

3.2 Cas de base

Cette section explique le cas de base de la procédure de découpage. Ce cas de base repose sur l'hypothèse que si nous ne pouvons traverser l'image ni verticalement ni horizontalement, il s'agit d'un symbole simple que l'on peut comparer avec notre alphabet. Nous devons cependant préciser la signification que nous donnons au terme *traverser*.

Définition 3.2.1 *Nous dirons d'une image qu'elle est traversée verticalement (resp. horizontalement) s'il existe une colonne (resp. ligne) de l'image ne contenant pas de pixel noir.*

Voici dès lors la façon dont nous allons traiter le cas de base: nous allons générer deux droites verticales au travers de notre matrice de 1 et de 0. Leur position initiale sera la moitié de l'image. Nous translaterons l'une d'elles vers la gauche et l'autre vers la droite. D'un coté comme de l'autre, nous nous arrêterons si l'image peut être traversée ou si nous l'avons parcourue entièrement.

Nous distinguerons alors trois cas:

1. le cas où l'image ne peut être traversée verticalement
2. le cas où l'image est traversée une fois verticalement
3. le cas où l'image est traversée deux fois verticalement

3.2.1 Le cas où l'image est traversée deux fois verticalement

Dans ce cas nous recréons trois nouvelles images en divisant l'image initiale suivant les droites qui la traversent. Nous appliquons ensuite le programme de centrage des images aux trois nouvelles images. Les quatre entiers renvoyés par ce programme vont nous servir à la localisation du bloc dans notre image globale.

3.2.2 Le cas où l'image est traversée une fois verticalement

Dans ce cas, nous ne recréons non plus trois, mais deux images en divisant l'image initiale suivant la droite qui la traverse. Nous appliquons ensuite le programme de centrage des images aux deux nouvelles images.

3.2.3 Le cas où l'image ne peut être traversée verticalement

Dans ce cas, nous allons essayer de traverser l'image horizontalement de manière similaire à celle expliquée pour le cas vertical. Nous distinguerons une fois de plus trois cas:

1. le cas où l'image ne peut être traversée horizontalement
2. le cas où l'image est traversée une fois horizontalement

3. le cas où l'image est traversée deux fois horizontalement

Les cas deux et trois se traiteront de manière analogue aux cas deux et trois verticaux (voir sous-sections 3.2.1 et 3.2.2).

3.2.4 Le cas où l'image ne peut être traversée horizontalement

Nous sommes alors dans le cas où l'image ne peut être traversée ni verticalement ni horizontalement, nous sommes donc selon nos hypothèses dans le cas où nous avons affaire à un symbole. Nous lancerons alors l'étape de reconnaissance.

3.2.5 Organigramme

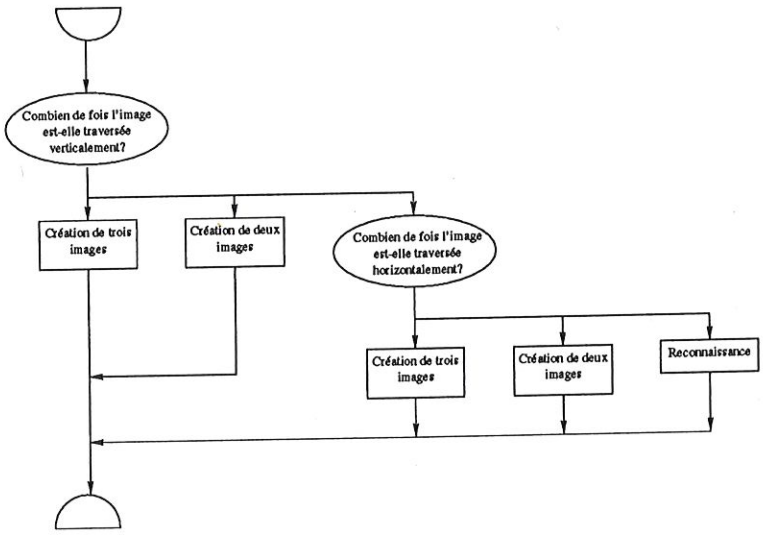

FIG. 3.1 – *Organigramme du cas de base.*

3.2.6 Prototypes de `traverse.cpp`

```
#define uint32 unsigned long
```

```
int traversverti(double **mat,uint32 nlig,uint32 ncol,int
  &vertiun,int &vertideux,int &ok1,int &ok2);
int travershoriz(double **mat,uint32 nlig,uint32 ncol,int
 &horizun,int &horizdeux,int &ok1,int &ok2);
```

Traversverti est la fonction utilisée pour le cas vertical et travershoriz celle pour le cas horizontal. Nous distinguerons,
les paramètres communs aux deux fonctions:

- `**mat` est un pointeur de type double, il désigne la matrice à traiter
- `nlig` (resp. `ncol`) est un entier long non signé contenant le nombre de lignes (resp. colonnes) de la matrice
- `ok1` (resp. `ok2`) est un entier. Après l'exécution de la fonction il vaut 1 si la première (resp. seconde) droite générée traverse la matrice et 0 dans le cas contraire.[2]

des paramètres propres à chacune d'entre elles:

- `vertiun` (resp. `vertideux`) est un entier. Après l'exécution de la fonction sa valeur est celle de la colonne par laquelle la première (resp. seconde) droite traverse la matrice.
- `horizun` (resp. `horizdeux`) est un entier. Après l'exécution de la fonction sa valeur est celle de la ligne par laquelle la première (resp. seconde) droite traverse la matrice.

3.2.7 Code de `traverse.cpp`

Pour le code du programme se référer au volume d'annexes.

2. Dans le cas où les deux droites traverseraient la matrice au même endroit, ok1 prend la valeur 0 et ok2 la valeur 1.

3.2.8 Prototypes de `creationimage.cpp`

Les fonctions creationimage sont celles utilisées lorsque nous connaissons les coupures afin de créer les nouvelles images. Nous distinguons une fois de plus deux types de fonctions:

- Les fonctions créant les nouvelles images en divisant la matrice de départ verticalement;
- Les fonctions créant les nouvelles images en divisant la matrice de départ horizontalement.

Parmi ces deux types, nous différencions les fonctions créant trois images, dans le cas où `ok1` et `ok2` prennent la valeur 1, et celles créant deux images, dans le cas où soit `ok1` soit `ok2` prend la valeur 1.

```
void imagehoriz(double **mat,char nom1[256],char nom2[256],char
nom3[256],uint32 nlig,uint32 ncol,int horizun,int horizdeux);
void imageverti(double **mat,char nom1[256],char nom2[256],char
nom3[256],uint32 nlig,uint32 ncol,int vertiun,int vertideux);
void imagehoriz(double **mat,char nom1[256],char nom2[256]
,uint32 nlig,uint32 ncol,int horizun);
void imageverti(double **mat,char nom1[256],char nom2[256]
,uint32 nlig,uint32 ncol,int vertiun);
```

Les fonctions créant les images par division verticale sont celles dont le nom est "imageverti". Celles créant deux images, sont celles ne prenant pas nom3 en paramètre.

- `**mat` est pointeur de type double désignant la matrice de départ.
- `nom1` (resp. `nom2` et `nom3`) est un chaine de caractères contenant le nom de la première (resp. deuxième et troisième (si nécessaire)) image créée
- `nlig` (resp. `ncol`) est un entier long non signé contenant le nombre de lignes (resp. colonnes) de la matrice de départ.

3.2.9 Code de `creationimage.cpp`

Pour le code du programme se référer au volume d'annexes.

3.2.10 La fonction reconnaissance

La fonction reconnaissance est celle à laquelle nous faisons appel lorsqu'il n'est possible de diviser la matrice d'aucune manière (verticale ou horizontale). La fonction compare l'image, ayant été préalablement remise à une taille de 64*64[3], avec toutes les images de notre alphabet. Le résultat est l'image pour laquelle la distance est maximale. Dans le cas où l'image n'est pas présente dans l'alphabet la fonction renvoit le nom de l'image de l'alphabet se rapprochant le plus de l'image à reconnaître au sens de notre distance[4].

3.2.11 Prototype de `reconnaissance.cpp`

```
char *reconnaissance(char nom[256]);
```

- `nom` est une chaîne de caractères contenant le nom de l'entité à reconnaître.

3.2.12 Code de `reconnaissance.cpp`

Pour le code du programme se référer au volume d'annexes.

3.2.13 Sauvegarde des résultats du cas de base

En vue de procéder au découpage totale de la formule, nous écrivons dans un fichier une série de résultats obtenus lors de l'exécution du cas de base.

3. Cette taille est la même que celle des éléments de notre alphabet. Nous expliquerons plus tard le choix de ces valeurs.

4. Voir la section 4.1 consacrée à la distance pour plus d'information

Une ligne de ce fichier comprend les informations suivantes:

- Une valeur entière nous informant sur "l'état" de l'image. Cette valeur vaut
  - 1 si l'image est une entité et a été reconnue
  - 2 si l'image est issue d'un découpage horizontal
  - 3 si l'image est issue d'un découpage vertical

  Remarque: Ces trois possibilités reprennent les trois cas possibles étant donné que l'étape de reconnaissance n'est lancée que si l'image ne peut plus être découpée et qu'elle donne toujours un résultat (l'élément de l'alphabet dont la distance à l'image reconnue est la plus grande).
- Une chaine de caractères contenant
  - le nom de l'image de notre dictionnaire obtenue par reconnaissance si la valeur entière est 1
  - le nom de l'image dans laquelle a été placé le résultat de la création d'image si la valeur entière est 2 ou 3
- Une chaîne de caractères contenant le nom de l'image sur laquelle le cas de base vient d'être appliqué
- Quatre valeurs entières permettant de localiser l'image dans l'image initiale de la formule. Grâce à ces quatre valeurs nous pouvons localiser les coin supérieur gauche et inférieur droit dans un repère dont l'origine serait le pixel inférieure gauche de l'image originale[5].

3.2.14 Exemples

Nous présentons maintenant trois exemples de l'exécution du cas de base. Chacun de ces exemples illustre une des possibilités du cas de base:

- reconnaissance
- découpage vertical

5. Les deux premières valeurs donnant les ordonnées des coins et les deux autres leurs abscisses

– découpage horizontal

Exemple 1

Si nous appliquons le cas de base à l'image "x.tif".
Ne pouvant traverser l'image ni verticalement ni horizontalement, le pro-

FIG. 3.2 – *Image de x.*

gramme exécute la reconnaissance sur l'image. Le programme ne crée pas de nouvelle image et inscrit les informations suivantes dans un fichier:

```
1 x.tif x.tif 0 0 0 0
```

FIG. 3.3 – *fichier résultat.*

Exemple 2

Nous appliquons maintenant le cas de base du découpage à la formule $x + y$.

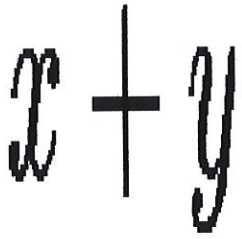

FIG. 3.4 – *Image de $x+y$.*

Nous parvenons à traverser l'image verticalement [6]

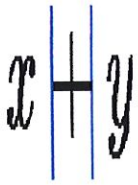

FIG. 3.5 – *Image du découpage de $x+y$.*

6. Les lignes bleues représentent les découpages.

de sorte qu'il créé les trois images suivantes:

FIG. 3.6 – *première image créée:* `formule1.tif`.

FIG. 3.7 – *deuxième image créée:* `formule2.tif`.

FIG. 3.8 – *troisième image créée:* `formule3.tif`.

Le programme inscrit les résultats suivants dans un fichier

```
3 formule1.tif xplusy.tif 103 30 1 28
3 formule2.tif xplusy.tif 127 20 48 85
3 formule3.tif xplusy.tif 103 0 103 128
```

FIG. 3.9 – *fichier résultat.*

Exemple 3

Le troisième exemple est celui illustrant le découpage horizontal de l'image. Pour ce faire nous appliquons le programme à la formule $\frac{x}{y}$.

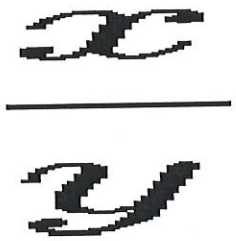

FIG. 3.10 – *image de* $\frac{x}{y}$.

Lors de l'exécution, nous n'arrivons pas à traverser l'image verticalement. Nous y parvenons horizontalement:

FIG. 3.11 – *image du découpage de* $\frac{x}{y}$.

Le programme crée les deux image suivantes:

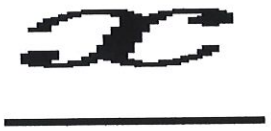

FIG. 3.12 – *première image créée:* `formule1.tif`.

FIG. 3.13 – *seconde image créée:* `formule2.tif`.

Et les résultats dans le fichier sont les suivants:

```
2 formule1.tif xdivy.tif 127 72 1 128
2 formule2.tif xdivy.tif 40 0 9 110
```

FIG. 3.14 – *fichier résultat.*

3.3 Cas général

Nous voyons dans les exemples 1 et 2 qu'après une seule application du cas de base, le découpage n'est pas complet:

- soit l'image créée par le cas de base n'est pas une entité (voir par exemple la figure 3.12)
- soit l'image est une entité et n'a pas encore été reconnue

Afin d'effectuer un découpage total de la formule et de reconnaître chaque entité nous allons parcourir le fichier résultat et réappliquer le cas de base lorsque l'image n'a pas été reconnue. Concrètement, nous lisons dans le fichier résultats, "l'état" et le nom de l'image.

- si "l'état" égale 1, nous passons à la ligne suivante dans le fichier
- si "l'état" égale 2 ou 3, nous réappliquons le cas de base à la formule dont nous avons lu le nom

Il faut remarquer que le fichier résultat se modifie au fur et à mesure. Nous nous arrêtons quand nous l'avons parcouru entièrement. En effet, nous sommes certains que nous n'arriverons à la fin du fichier qu'une fois que toutes les entités auront été reconnues, car lorsque nous rencontrons une image non reconnue le nombre de lignes du fichier augmente.

3.3.1 Organigramme

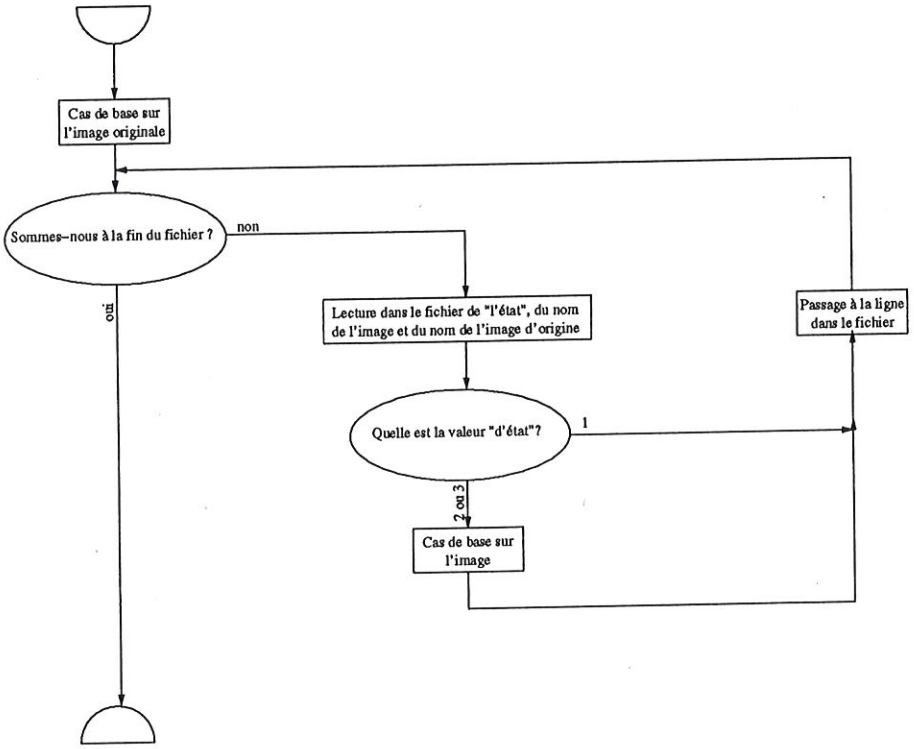

FIG. 3.15 – *Organigramme du cas général.*

3.3.2 Prototype de `creationarbre.cpp`

```
void creationarbre(char nom[256]);
```

- `nom` est une chaine de caractères contenant le nom de l'image devant être découpée.

3.3.3 Exemple

Nous allons maintenant détailler un exemple un peu plus complexe étape par étape. Ce que nous entendons par exemple plus complexe est un exemple nécessitant plus d'une application du cas de base. L'exemple choisi est la formule

$$\frac{x * y}{x + y}$$

.

$$\frac{x * y}{x + y}$$

FIG. 3.16 – *Image de la formule* $\frac{x*y}{x+y}$.

Première étape

Nous appliquons une première fois le cas de base à l'image de la formule. Cette image peut être traversée horizontalement:

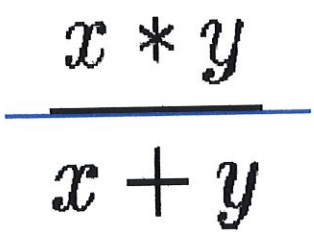

FIG. 3.17 – *Image du découpage de la formule* $\frac{x*y}{x+y}$.

les deux images suivantes sont créées:

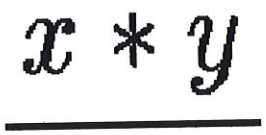

FIG. 3.18 – *première image créée:* `formule1.tif`.

FIG. 3.19 – *seconde image créée:* `formule2.tif`.

et le fichier résultat est de la forme:

```
2 formule1.tif complex.tif 127 71 1 128
2 formule2.tif complex.tif 49 0 2 125
```

FIG. 3.20 – *fichier résultat.*

deuxième étape

Nous lisons dans la première ligne du fichier "l'état", le nom de l'image et le nom de l'image d'origine. "L'état" étant différent de 1, nous appliquons le cas de base à `formule1.tif` (voir figure 3.18).
L'image peut être traversée horizontalement.

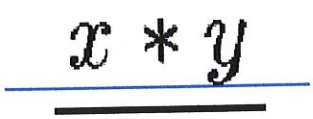

FIG. 3.21 – *Image du découpage de* `formule1.tif`.

Les deux images suivantes sont créées:

FIG. 3.22 – *première image créée:* `formule4.tif`.

FIG. 3.23 – *seconde image créée:* `formule5.tif`.

et le fichier résultat devient:

```
2 formule1.tif complex.tif 127 71 1 128
2 formule2.tif complex.tif 49 0 2 125
2 formule4.tif formule1.tif 127 87 11 118
2 formule5.tif formule1.tif 73 71 2 129
```

FIG. 3.24 – *fichier résultat.*

Troisième étape

Nous lisons la deuxième ligne du fichier. L'état est différent de 1, nous appliquons le cas de base à `formule2.tif` (voir figure 3.19). L'image peut être traversée verticalement:

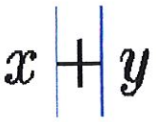

FIG. 3.25 – *Image du découpage de* `formule2.tif`.

les trois images suivantes sont créées:

FIG. 3.26 – *première image créée:* `formule7.tif`.

FIG. 3.27 – *deuxième image créée:* `formule8.tif`.

FIG. 3.28 – *troisième image créée:* `formule9.tif`.

et le fichier résultat devient:

```
2 formule1.tif complex.tif 127 71 1 128
2 formule2.tif complex.tif 49 0 2 125
2 formule4.tif formule1.tif 127 87 11 118
2 formule5.tif formule1.tif 73 71 2 129
3 formule7.tif formule2.tif 39 12 3 29
3 formule8.tif formule2.tif 49 8 48 84
3 formule9.tif formule2.tif 39 0 102 126
```

FIG. 3.29 – *fichier résultat.*

Fin de l'exécution

Le programme continue en effectuant un découpage vertical de `formule4.tif`. Ensuite chacune des entités est reconnue.

Arbre de l'exemple

Nous présentons ici un arbre représentant le découpage effectué par le programme sur la formule $\frac{x*y}{x+y}$:

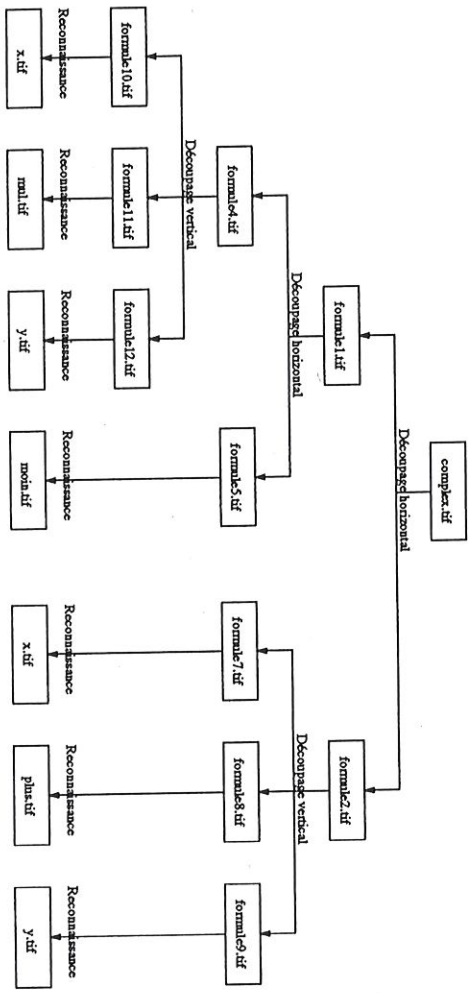

FIG. 3.30 – *Arbre du découpage.*

3.4 Critiques du découpage

Le découpage ne prend pas en compte tous les cas. En effet, dans le cas de caractères emboités, ou d'entités que nous pouvons encore découper mais qui ont déjà été reconnues.

3.4.1 Les caractères emboités

L'exemple de caractères emboités posant problème sont les racines et les intégrales avec leurs bornes. Le problème vient du fait que, ne pouvant être traversée ni verticalement ni horizontalement, l'image est considérée comme une entité.

Exemples

Si nous appliquons le programme de découpage à la formule $\sqrt{x+y}$:

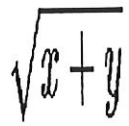

FIG. 3.31 – *Image de* $\sqrt{x+y}$.

L'image ne peut être traversée bien que n'étant pas une entité. Le programme applique la reconnaissance qui identifie l'image à l'élément représentant $\int$ de notre alphabet:

```
1 int.tif racine.tif 0 0 0 0
```

FIG. 3.32 – *fichier résultat.*

Un autre exemple est celui d'une intégrale et de ses bornes:

FIG. 3.33 – *Image de* $\int_b^a$.

Le programme effectue un découpage vertical et crée deux images:

FIG. 3.34 – *première image créée:* `formule1.tif`.

FIG. 3.35 – *seconde image créée:* `formule2.tif`.

`formule1.tif` ne peut plus être traversée bien que n'étant pas une entité. Le programme reconnaît l'image comme étant une intégrale mais ne reconnait pas la borne inférieure.

```
3 formule1.tif integrale.tif 125 0 1 87
3 formule2.tif integrale.tif 127 110 97 128
1 int.tif formule1.tif 125 0 1 87
1 a.tif formule2.tif 127 110 97 128
```

FIG. 3.36 – *fichier résultat.*

3.4.2 Les entités découpables

Certaines entités peuvent être découpées alors qu'elles devraient être reconnues. C'est le cas par exemple pour:

- =
- i
- j
- ...

Nous avons modifié le programme de découpage pour solutionner ce problème. Avant de voir si l'image peut-être traversée horizontalement, nous calculons la distance entre l'image et l'élément de notre alphabet correspondant au "=". Si cette distance est suffisamment grande, nous considérons que nous avons reconnu un "=". Sinon nous appliquons le découpage vertical.

Remarque: Nous avons essayé une solution similaire pour le cas du "j" et du "i", mais le taux de reconnaissance n'est pas suffisant pour ces lettres.

3.4.3 Le nombre de découpages

Le nombre de découpage est fixe. En effet, pour chaque étape (différente de la reconnaissance) nous effectuons un nombre fixe de découpages, deux ou trois, et ce même si l'image peut-être découpée plus que deux ou trois fois. Il arrive également que nous n'effectuions que deux découpages lorsque nous devrions en effectuer trois. Nous pourrions imaginer un nombre de découpage variable de sorte qu'à chaque étape tous les découpages soient effectués. Cette façon de travailler faciliterait le regroupement des différentes entités. En effet cela éviterait de séparer les entités d'un même niveau dans une formule.

Chapitre 4

Distance et alphabet

Nous avons mentionné à plusieurs reprises les termes *distance* et *alphabet*, nous allons maintenant développer ces deux aspects de notre travail.

4.1 Distance

La distance que nous utilisons dans ce travail est basée sur une distance déjà utilisée par Alexandra Dessy et Marie-Pierre Fivet dans leur mémoire [4]. Nous travaillons avec des images binaires [1]. La distance entre deux images binaires f et g est la distance définie comme suit. Supposons que les fonctions $f(j)$ et $g(j)$ prennent la valeur 1 si le pixel j est blanc et la valeur 0 sinon (i.e. le pixel j est noir), alors

$$d(f,g) = \sum_{j \in npixel} \delta_{f(j),g(j)}/npixel$$

où $npixel$ est le nombre de pixels et où

$$\delta_{f(j),g(j)} = \begin{cases} 1 \; si \; f(j) = g(j) \\ 0 \; sinon \end{cases}$$

1. une image binaire est une image dont les pixels sont soit noir soit blanc

La distance ainsi définie représentera donc la proportion de pixels identiques d'une image à l'autre, contrairement à celle utilisée dans le mémoire [4] qui est le nombre et non la proportion de pixels communs.

4.2 Alphabet

L'alphabet est composé des entités étalons ou "training sets". C'est à ces entités étalons que sont comparées les entités observées dans l'étape de découpage. Nous avons écrit un programme qui crée cet alphabet. Les éléments de notre alphabet sont des images TIFF de taille 64*64 pixels. Cette taille a été choisie pour la pertinence de la distance que nous utilisons. En effet, il nous fallait choisir un compromis entre un nombre suffisant de pixels pour que notre distance ait un sens: celle-ci étant basée sur la proportion de pixels en commun entre deux images, il fallait que ce nombre de pixels soit suffisant pour que la distance soit significative. Et un nombre de pixels pas trop important pour que l'étape de remise à l'échelle n'altère pas trop la forme de l'élément. Cet alphabet est composé:

- des chiffres;
- des lettres de l'alphabet usuel en minuscule;
- de certaines lettres de l'alphabet grec;
- des opérateurs mathématiques: +, -,=, *, /, $\int$, $\sum$.

Les éléments de notre alphabet sont générés en LaTeX. Cet alphabet n'est pas exhaustif mais pourrait être completé sans trop de difficultés: il suffit d'ajouter au fichier `dictionnaire.txt` l'expression LaTeX correspondant à l'élément à ajouter à l'alphabet et au fichier `nom1.txt` le nom sous lequel l'image de cet élément sera créée. Il va de soi qu'aucun résultat de notre reconnaissance ne peut être absent de notre alphabet.

4.2.1 Création de l'alphabet

Nous utilisons donc un programme pour créer l'alphabet. Ce programme lit dans un fichier (`dictionnaire.txt`) les expressions à générer et dans un autre (`nom1.txt`) les noms sous lesquels les images vont être sauvées. Il utilise le script `script.sh` pour générer l'image TIFF. Il applique ensuite les programmes de pré-traitement pour entre autre supprimer les contours blancs et remettre l'image à la taille désirée.

Remarque: il faut remarquer que l'étape de reconnaissance étant basée sur le nom des images, il est donc impératif que la correspondance entre l'expression et l'image soit sans équivoque.

Code de `dictionnaire.cpp`

Pour le code du programme se référer au volume d'annexes.

4.2.2 `script.sh`

Le fichier `script.sh` est utilisé lors de la création de l'alphabet, il sera encore utilisé dans une étape ultérieure de notre travail. Il a pour but de compiler un fichier LaTeX et d'effectuer une capture d'écran sur l'image créée. Ce script permet d'automatiser l'étape de création d'image de formule et de pouvoir faire appel à cette étape lors de l'exécution de nos programmes $C++$[2].

Déscription du script

```
cat essai_debut.tex > essai.tex
cat formule.tex >> essai.tex
cat essai_fin.tex >> essai.tex
```

2. En utilisant la fonction `system` du langage $C++$.

```
latex essai.tex > /dev/null
dvips -q -T200pt,100pt essai.dvi -o essai.ps
gs -dNOPAUSE -r300 -dBATCH -sOutputFile=imagetemp.tif
-sDEVICE=tiffg3 essai.ps > /dev/null
```

Les quatre premières lignes du script ont pour but de concaténer les trois fichiers .tex et de compiler le fichier `essai.tex`. L'expression de la formule à générer se trouve dans `formule.tex`. La ligne suivante a pour but de transformer le fichier `dvi` résultat de la compilation LaTeX en un fichier `postscript`. Les dernières lignes ont pour but de créer une image TIFF sur base du fichier `postscript`. Il faut ensuite procéder au pré-traitement de l'image pour notamment remettre l'image à la bonne taille.

4.3 Distance entre nos entités étalons

Nous avons décidé de calculer la distance entre certains éléments de notre alphabet. Ceci afin de pouvoir nous rendre compte de la ressemblance entre certains éléments de notre alphabet. Nous avons généré ces étalons dans les tailles de police de caractère de 11 pts et de 12 pts. Nous avons ensuite calculé la distance entre chaque élément de l'alphabet.

4.3.1 Première essai avec des images de 16*16 pixels

Lors d'une première étape nous avions fixé la taille des éléments de notre alphabet à 16*16 pixels. Le tableau 4.1, où l'élément en minuscule représente l'image en 11 pts et celui en majuscule celle en 12 pts[3], représente les résultats obtenus par la distance. Nous ne sommes pas satisfait des résultats. En effet, nous pouvons voir que la distance entre un "e" en 11 pts et un "c" dans la même taille est de 0.867188 pour une distance de 0.804688 entre ce même "e" et un "e" en 12 pts. Le "e" en 11pts ressemble donc plus à un "c" qu'à un autre "e". Augmentons la taille des images de notre alphabet.

3. Excepté pour le alpha où α représente le alpha en 11 pts et $\alpha 2$ celui en 12 pts

	x	X	e	E	c	C	f	F
x	1	0.746094	0.507812	0.445312	0.546875	0.457031	0.589844	0.546875
X	–	1	0.488281	0.410156	0.519531	0.445312	0.6875	0.707031
e	–	–	1	0.804688	0.867188	0.792969	0.464844	0.429688
E	–	–	–	1	0.796875	0.964844	0.441406	0.398438
c	–	–	–	–	1	0.832031	0.441406	0.414062
C	–	–	–	–	–	1	0.453125	0.417969
f	–	–	–	–	–	–	1	0.871094
F	–	–	–	–	–	–	–	1

	a	A	α	$\alpha 2$	int	INT	sum	SUM
x	0.558594	0.457031	0.433594	0.523438	0.519531	0.523438	0.511719	0.476562
X	0.554688	0.523438	0.484375	0.488281	0.617188	0.636719	0.539062	0.542969
e	0.644531	0.597656	0.660156	0.523438	0.441406	0.453125	0.511719	0.5
E	0.542969	0.628906	0.613281	0.570312	0.417969	0.40625	0.488281	0.484375
c	0.660156	0.683594	0.628906	0.539062	0.425781	0.4375	0.566406	0.539062
C	0.515625	0.617188	0.578125	0.535156	0.429688	0.433594	0.523438	0.519531
f	0.359375	0.265625	0.398438	0.402344	0.835938	0.832031	0.539062	0.550781
F	0.339844	0.285156	0.371094	0.351562	0.839844	0.867188	0.574219	0.578125
a	1	0.828125	0.710938	0.605469	0.289062	0.277344	0.382812	0.386719
A	–	1	0.734375	0.660156	0.234375	0.253906	0.40625	0.433594
α	–	–	1	0.730469	0.367188	0.363281	0.429688	0.410156
$\alpha 2$	–	–	–	1	0.425781	0.421875	0.480469	0.476562
int	–	–	–	–	1	0.925781	0.625	0.644531
INT	–	–	–	–	–	1	0.636719	0.671875
sum	–	–	–	–	–	–	1	0.832031
SUM	–	–	–	–	–	–	–	1

TAB. 4.1 – *Tableau des distances entre des images 16*16 pixels.*

4.3.2 Second essai avec des images de 64*64 pixels

Le tableau 4.2 reprend les mêmes renseignements mais pour des images d'une taille de 64*64. Les résultats nous satisfont davantage, même si certains problèmes persistent, car le taux de reconnaissance entre deux même lettres (dans des tailles différentes) est plus important. Nous avons encore essayé avec des images de taille supérieure mais la mise à l'échelle nous empéchait alors de distinguer les entités-étalons. Nous garderons donc la taille de 64*64 pour les éléments de notre alphabet. En plus de nous informer sur notre distance, ce tableau va permettre de complèter la matrices d'alternatives[4] en y ajoutant les éléments pour lesquels la distance dépasse un certain seuil.

4. Voir section 5.1.2.

	x	X	e	E	c	C	f	F
x	1	0.882812	0.520264	0.54248	0.531738	0.572021	0.741943	0.736328
X	–	1	0.505615	0.532227	0.513184	0.55835	0.781494	0.762695
e	–	–	1	0.878174	0.856689	0.83252	0.469727	0.477295
E	–	–	–	1	0.812988	0.925049	0.516357	0.526367
c	–	–	–	–	1	0.845947	0.501221	0.511719
C	–	–	–	–	–	1	0.522461	0.532471
f	–	–	–	–	–	–	1	0.948975
F	–	–	–	–	–	–	–	1

	a	A	α	$\alpha 2$	int	INT	sum	SUM
x	0.417236	0.466309	0.498047	0.526855	0.693848	0.69873	0.575684	0.574463
X	0.417236	0.450195	0.497559	0.536621	0.735352	0.737793	0.615234	0.611084
e	0.688477	0.639893	0.614014	0.595947	0.501709	0.501221	0.561768	0.555176
E	0.713623	0.691895	0.644043	0.631348	0.556152	0.556152	0.600098	0.592529
c	0.69458	0.607422	0.565918	0.562012	0.532715	0.53418	0.600098	0.596436
C	0.725586	0.71167	0.648682	0.645752	0.565674	0.567139	0.611084	0.608398
f	0.427734	0.466064	0.49292	0.528564	0.881592	0.883057	0.640381	0.643555
F	0.444092	0.478027	0.500977	0.540039	0.884277	0.890625	0.660645	0.660889
a	1	0.837646	0.744873	0.696045	0.461182	0.462646	0.502197	0.499512
A	–	1	0.769531	0.768555	0.486816	0.492188	0.53418	0.528564
α	–	–	1	0.899902	0.518066	0.521973	0.529785	0.527588
$\alpha 2$	–	–	–	1	0.558594	0.562988	0.556641	0.55835
int	–	–	–	–	1	0.977539	0.69043	0.696045
INT	–	–	–	–	–	1	0.688965	0.69458
sum	–	–	–	–	–	–	1	0.957275
SUM	–	–	–	–	–	–	–	1

TAB. 4.2 – *Tableau des distances entre des images 64*64 pixels.*

Chapitre 5

Regroupement des entités

Une fois l'arbre créé sur base de l'image, nous devons transformer celui ci en une expression LaTeX correspondant à la formule dont nous avons découpé l'image. Cette étape de regroupement des entités observées se fait au moyen de `concatenation.cpp`. Son utilisation se fait de manière récursive. Nous allons expliquer le cas de base.

5.1 Cas de base

Le cas de base reprend les cas simples d'une entité reconnue, d'un découpage vertical (à une ou deux coupures), d'un découpage horizontale (à une ou deux coupures). Nous différencions les cas grâce à la variable "état" présente dans le fichier issu de la fonction `creationarbre.cpp`.

5.1.1 Passage d'une entité reconnue à une expression

Cette étape est effectuée par le sous-programme `expression.cpp`. Pour une entité reconnue, le programme écrit dans un fichier l'expression associée à l'image de l'alphabet que le programme de reconnaissance a associé à l'entité. Etant donné que cette association se fait sur base du nom de l'image, nous rappelons ici l'importance du fait que la création de l'alphabet se fasse sans

équivoque. En plus de l'expression reconnue, le programme ajoute dans le fichier les alternatives possibles à cette expression.

Prototype de `expression.cpp`

```
void expression(char nom[256],char nom2[256]);
```

- `nom` est une chaîne de caractères contenant le nom de l'entité étalon dont nous devons donner l'expression.
- `nom2` est une chaîne de caractères contenant le nom du fichier dans lequel vont être placées l'expression et les éventuelles alternatives.

Code de `expression.cpp`

Pour le code du programme se référer au volume d'annexes.

5.1.2 Alternatives

Le fait de considérer les alternatives permet de diminuer les problèmes de mauvaise reconnaissance ou d'exponentiation et d'indiçage. En effet si pour le cas de formules générées de la même manière que notre alphabet[1], le taux de bonne reconnaissance est très satisfaisant, il se pourrait que pour des formules d'autres origines (autres logiciels ou manuscrites), ce taux soit moins acceptable. La matrice d'alternatives a pour but de proposer une solution dans le cas d'une mauvaise reconnaissance. Cette matrice contient pour chaque élément de notre alphabet les éléments qui lui sont proches.

La matrice que nous utilisons est construite de la façon suivante: pour chaque élément de l'alphabet nous inscrivons dans le fichier `alternative.txt` l'expression associée à l'élément suivie du nombre d'alternatives et, si ce

1. au moyen de LaTeX

dernier est différent de 0, les alternatives proposées pour l'expression. Par exemple:

```
a 1 \alpha
```

signifie que la seule alternative proposée pour le caractère "a" est α.

5.1.3 Regroupement lors de découpages verticaux

Lors d'un découpage vertical nous avons deux possibilités:

1. le cas d'une coupure
2. le cas de deux coupures

Cas d'une coupure

Nous avons donc deux images issues du découpage vertical. Nous appliquons le programme de concaténation à ces deux images et nous obtenons deux fichiers:

- un contenant l'expression et les alternatives correspondant à l'image de gauche
- un contenant l'expression et les alternatives correspondant à l'image de droite.

Nous concaténons les expressions des deux fichiers en effectuant toutes les combinaisons possibles: chaque élément du fichier associé à l'image de droite est concaténé avec chaque élément du fichier associé à l'image de gauche. D'une telle façon, si le premier fichier comprend n éléments et le second m éléments, le fichier contenant les concaténations comprend $n \times m$ éléments.

Cas de deux coupures

Le cas de deux coupures se règle de façon similaire excepté que nous disposons d'un fichier en plus étant donné que le découpage a créé trois images.

C'est à cette étape que nous réglons le problème des exposants et des indices. En effet pour les éléments dont nous avons jugé qu'ils étaient fréquement utilisés en exposant ou en indice, nous avons ajouté dans la matrice d'alternatives l'expression précédée d'un "^" (dans le cas d'un exposant) et d'un "_" (dans le cas d'un indice) qui sont les expressions LaTeX pour générer les exposants et les indices.

5.1.4 Regroupement lors de découpages horizontaux

Nous distinguons également le cas d'une seule coupure de celui de deux coupures. Nous développerons d'abord celui de deux coupures.

Cas de deux coupures

Lors de deux coupures, trois images sont créées. Nous regardons si l'image du milieu est reconnue comme un "-"[2]:

- si c'est le cas nous avons une fraction. Nous créons alors les combinaisons générant une fraction avec pour numérateur les éléments du fichier d'expressions obtenus en appliquant le programme de concaténation à l'image du dessus et pour dénominateur ceux du fichier obtenu en appliquant le programme de concatenation à l'image du dessous.
- dans le cas contraire, nous considérons que les éléments du fichier d'expressions obtenu grâce à l'image du dessus sont exposants et ceux de celui obtenu grâce à l'image du dessous sont indices des éléments du fichier d'expressions relatif à l'image du milieu. Ce serait le cas par exemple pour une somme avec ses bornes inférieures et supérieures de part et d'autre du symbole $\sum$.

Cas d'une coupure

Pour le cas d'une coupure nous posons l'hypothése suivante:

2. En effet pour notre alphabet, le "-" et la barre de fraction sont équivalent à une image de 64*64 pixels entièrement noire.

Hypothèse: *Si l'une des deux images créées lors de l'étape de découpage peut encore être découpée horizontalement nous sommes dans le cas d'une fraction.*

En accord avec cette hypothèse nous fonctionnons de la sorte: nous regardons les deux images créées par le découpage. Nous distinguons alors trois possibilités:

- l'image du **dessus** peut être redécoupée: nous prenons alors les expressions obtenues en appliquant le programme de concaténation à l'image du dessus comme numérateur de notre fraction et celles obtenues grâce à l'image du dessous comme dénominateur.
- l'image du **dessous** peut être redécoupée: nous prenons alors les expressions obtenues en appliquant le programme de concaténation à l'image du dessous comme dénominateur et celles obtenues grâce à l'image du dessus comme numérateur.
- aucune des deux images ne peut être découpée: nous considérons donc que les expressions obtenues en appliquant le programme de concaténation à l'image du dessus est en exposant et celles obtenues grâce à l'image du dessous en indice, un peu comme dans le cas de deux coupures où nous ne retrouvions pas de barre de fraction.

5.2 Prototype de `concatenation.cpp`

```
int concatenation(char nom[256],char nom2[256],ifstream &nomexpr);
```

- `nom` est une chaîne de caractères contenant le nom de l'élément de l'alphabet auquel nous avons associé l'entité.
- `nom2` est une chaîne de caractères contenant le nom du fichier `.txt` dans lequel l'expression et les alternatives vont être placées
- `nomexpr` est un fichier contenant les noms des fichiers `txt` temporaires.

5.3 Code de `concatenation.cpp`

Pour le code du programme se référer au volume d'annexes.

5.4 Générations et comparaisons

Nous disposons maintenant d'un fichier contenant les expressions et leurs altenatives, il nous faut donc décider de la formule que nous garderons. Pour ce faire nous allons générer les images correspondant aux expressions et calculer la distance entre ces images et l'image de départ que nous devions reconnaître. Une fois de plus nous utiliserons `script.sh`. Les résultats seront placés dans un fichier ultime `resultat.txt`. DAns un premier temps,cette étape de la procédure nécessitait une intervention de l'utilisateur: en effet, certaines des expressions que nous avons créées ne sont pas compilables en LaTeX pour des erreurs de syntaxe évidentes. Par exemple:

```
^2+_3
```

Pour chaque expression du fichier il était donc demandé à l'utilisateur s'il voulait générer l'image correspondant. C'était donc à l'utilisateur d'éviter tout problème de compilation LaTeX. Dans un second temps, nous avons remarqué que cette étape était fastidieuse et nous avons décidé de l'automatiser.

5.5 Nombre d'expressions

En utilisant le programme nous remarquons que le nombre d'expressions est très grand. En effet si le découpage nous donne k entités avec n_i le nombre d'alternatives pour l'entité i, le nombre d'expressions dans le fichier résultat vaut $\prod_{i=1}^{k} n_i$. Ce nombre devient vite important, d'autant plus qu'une part non négligeable d'alternatives conduit à des erreurs de compilation LaTeX. Afin de limiter ce nombre, nous avons modifié quelque peu le programme `expression.cpp`: lors de l'étape où nous ajoutions systématiquement les

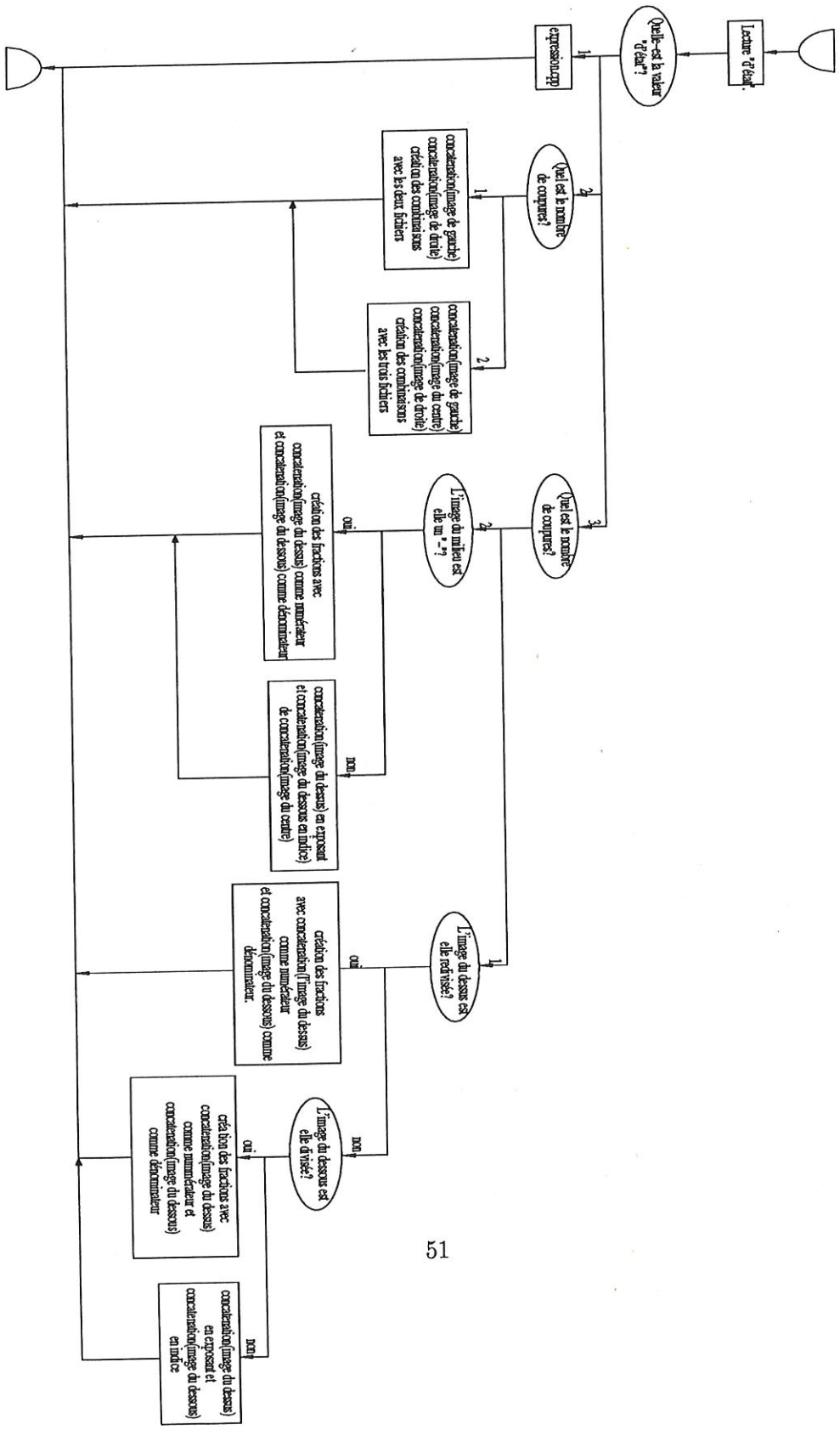

FIG. 5.1 – *Organigramme de* `concatenation.cpp`.

alternatives au fichier, nous demandons maintenant à l'utilisateur s'il désire l'ajouter. Nous diminuons de telle sorte le nombre d'alternatives à tester.

Chapitre 6

Applications

Nous allons appliquer notre programme sur certaines formules générées en LaTeX ainsi que sur des formules manuscrites ou issues de Word. Nous n'éliminerons que les alternatives succeptibles de poser problème lors de la compilation. Ce que nous présentons ici sont l'expressions de la formule et le fichier résultat final.

6.1 Formules LaTeX

Utilisons d'abord le programme sur la formule:

$$a^2 + b^2 = y$$

La figure 6.1 reprend les résulats.

```
a2+b2=y 0.842712        \alpha2+b2=y 0.833313
a2+b^2=y 0.92572        \alpha2+b^2=y 0.890808
a2+b_2=y 0.841248       \alpha2+b_2=y 0.826233
a^2+b2=y 0.929504       \alpha^2+b2=y 0.92572
a^2+b^2=y 1             \alpha^2+b^2=y 0.926941
a^2+b_2=y 0.97876       \alpha^2+b_2=y 0.913452
a_2+b2=y 0.828918       \alpha_2+b2=y 0.84259
a_2+b^2=y 0.97699       \alpha_2+b^2=y 0.917236
a_2+b_2=y 0.856018      \alpha_2+b_2=y 0.84259
```

FIG. 6.1 – *fichier résultat.*

Nous remarquons que l'expression pour laquelle la distance est la plus importante correspond à l'expression pour générer notre première image test. La formule a donc été correctement reconnue. Utilisons maintenant notre programme sur la formule plus compliquée:

$$\frac{a^2 + b^2}{25} = y$$

La figure 6.2 reprend les résulats.

```
\frac{a2+b2}{25}=y 0.876648
\frac{a2+b2}{2^5}=y 0.871521
\frac{a2+b2}{2_5}=y 0.867371
\frac{a2+b^2}{25}=y 0.953186
\frac{a2+b^2}{2^5}=y 0.919006
\frac{a2+b^2}{2_5}=y 0.886841
\frac{a2+b_2}{25}=y 0.882812
\frac{a2+b_2}{2^5}=y 0.873413
\frac{a2+b_2}{2_5}=y 0.869385
\frac{a^2+b2}{25}=y 0.958923
\frac{a^2+b2}{2^5}=y 0.92511
\frac{a^2+b2}{2_5}=y 0.889404
\frac{a^2+b^2}{25}=y 1
\frac{a^2+b^2}{2^5}=y 0.958435
\frac{a^2+b^2}{2_5}=y 0.909241
\frac{a^2+b_2}{25}=y 0.989563
\frac{a^2+b_2}{2^5}=y 0.947937
\frac{a^2+b_2}{2_5}=y 0.901733
\frac{a_2+b2}{25}=y 0.881104
\frac{a_2+b2}{2^5}=y 0.871704
\frac{a_2+b2}{2_5}=y 0.867371
\frac{a_2+b^2}{25}=y 0.988037
\frac{a_2+b^2}{2^5}=y 0.94635
\frac{a_2+b^2}{2_5}=y 0.900085
\frac{a_2+b_2}{25}=y 0.89209
\frac{a_2+b_2}{2^5}=y 0.882141
\frac{a_2+b_2}{2_5}=y 0.872009
```

```
\frac{\alpha2+b2}{25}=y 0.869995
\frac{\alpha2+b2}{2^5}=y 0.866882
\frac{\alpha2+b2}{2_5}=y 0.86554
\frac{\alpha2+b^2}{25}=y 0.932739
\frac{\alpha2+b^2}{2^5}=y 0.903381
\frac{\alpha2+b^2}{2_5}=y 0.87616
\frac{\alpha2+b_2}{25}=y 0.869568
\frac{\alpha2+b_2}{2^5}=y 0.863342
\frac{\alpha2+b_2}{2_5}=y 0.861206
\frac{\alpha^2+b2}{25}=y 0.952942
\frac{\alpha^2+b2}{2^5}=y 0.923035
\frac{\alpha^2+b2}{2_5}=y 0.889404
\frac{\alpha^2+b^2}{25}=y 0.954102
\frac{\alpha^2+b^2}{2^5}=y 0.919739
\frac{\alpha^2+b^2}{2_5}=y 0.889343
\frac{\alpha^2+b_2}{25}=y 0.948547
\frac{\alpha^2+b_2}{2^5}=y 0.914124
\frac{\alpha^2+b_2}{2_5}=y 0.884338
\frac{\alpha_2+b2}{25}=y 0.878357
\frac{\alpha_2+b2}{2^5}=y 0.872192
\frac{\alpha_2+b2}{2_5}=y 0.868225
\frac{\alpha_2+b^2}{25}=y 0.951721
\frac{\alpha_2+b^2}{2^5}=y 0.917236
\frac{\alpha_2+b^2}{2_5}=y 0.887756
\frac{\alpha_2+b_2}{25}=y 0.883728
\frac{\alpha_2+b_2}{2^5}=y 0.873901
\frac{\alpha_2+b_2}{2_5}=y 0.870667
```

FIG. 6.2 – *fichier résultat.*

Une fois de plus la reconnaissance ne pose pas de problème. Utilisons le programme sur la formule:

$$\frac{a^2}{2}+\frac{b^2}{5}=y$$

La figure 6.3 reprend les résulats.

```
\frac{a2}{2}+\frac{b2}{5}=y 0.894592
\frac{a2}{2}+\frac{b^2}{5}=y 0.949402
\frac{a2}{2}+\frac{b_2}{5}=y 0.89209
\frac{a^2}{2}+\frac{b2}{5}=y 0.958252
\frac{a^2}{2}+\frac{b^2}{5}=y 1
\frac{a^2}{2}+\frac{b_2}{5}=y 0.989136
\frac{a_2}{2}+\frac{b2}{5}=y 0.893799
\frac{a_2}{2}+\frac{b^2}{5}=y 0.989624
\frac{a_2}{2}+\frac{b_2}{5}=y 0.908203
\frac{\alpha2}{2}+\frac{b2}{5}=y 0.88208
\frac{\alpha2}{2}+\frac{b^2}{5}=y 0.924011
\frac{\alpha2}{2}+\frac{b_2}{5}=y 0.878174
\frac{\alpha^2}{2}+\frac{b2}{5}=y 0.954712
\frac{\alpha^2}{2}+\frac{b^2}{5}=y 0.94928
\frac{\alpha^2}{2}+\frac{b_2}{5}=y 0.943054
\frac{\alpha_2}{2}+\frac{b2}{5}=y 0.895142
\frac{\alpha_2}{2}+\frac{b^2}{5}=y 0.946777
\frac{\alpha_2}{2}+\frac{b_2}{5}=y 0.892578
```

FIG. 6.3 – *fichier résultat.*

avec une bonne reconnaissance.

6.2 Formules d'autres origines

Nous reprenons maintenant la formule

$$\frac{a^2 + b^2}{25} = y$$

Cette formule sera générées de trois façon différentes:

1. en LaTeX
2. en Word
3. manuscrite

Les résultats dans le cas où la formule est générée en LaTeXse trouve plus haut.

Pour la formule venant de Word (voir figure 6.4), les résultats ne sont pas satisfaisants. Le programme donne comme formule la plus proche:

`\frac{f+_2}{2s}=/`

avec une distance de 0.900635

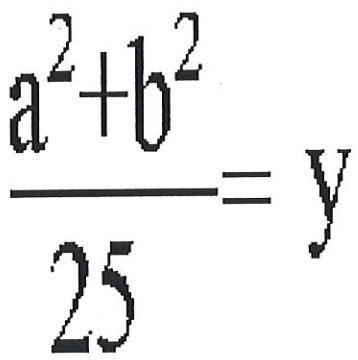

FIG. 6.4 – *formule venant de Word.*

Mais il semble que les erreurs soient plutôt dues à l'étape de reconnaissance qu'à celle du découpage. Il faudrait voir si nous pouvons passer outre de ces erreurs en modifiant la matrice d'alternatives.

Pour la formule manuscrite le pogramme ne donne aucun résultat. Nous supposons que cela est dû à la mauvaise numérisation de notre image (voir figure 6.5).

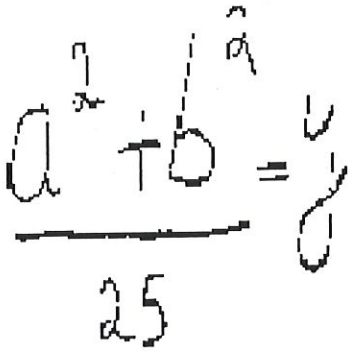

FIG. 6.5 – *formule manuscrite.*

Conclusion et pistes de réflexions.

Conclusion

Nous avons appliqué notre programme sur divers exemples. Il semble que pour des formules générées en LaTeX les résultats soient convaincants. Pour les formules d'autres origines, il faudrait voir en modifiant la matrice d'alternatives. Enfin, pour les formules manuscrites, l'étape de numérisation n'étant pas au point nous ne pouvons tirer aucune conclusion.
Notre méthode pourrait faire l'objet de recherches supplémentaires qui sont détaillées dans la section suivante.

Pistes de réflexion

Traitement des images

Nous avons expliqué dans un chapitre toutes les traitements succeptibles d'être faits sur une image lors de sa numérisation. Il va de soi que tous ces traitements devront être envisagés si le mode d'acquisition des images change.

Le découpage

Comme nous l'avons déjà expliqué le découpage ne répond pas à toutes les attentes: nous rencontrons des problèmes pour les caractères emboités et les entités découpables. Une partie de ces problèmes ont été surmontés. Il reste cependant certains cas problematiques. Une solution envisageable pour le découpage de caractères emboités serait un découpage suivant la diagonale: Le principe serait le suivant: les deux droites rouges parcourraient

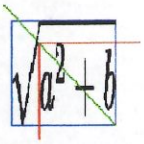

FIG. 6.6 – *Découpage selon la diagonale.*

l'image (une verticalement et l'autre horizontalement. Si les deux arrivent à rejoindre la diagonale (droite verte) sans rencontrer de pixel noir, nous pouvons effectuer le découpage. Un des problèmes serait de trouver la bonne inclinaison pour la diagonale.

La reconnaissance

Notre travail est plus basé sur le découpage et le regroupement selon la structure particulière des formules mathématiques. En ce qui concerne la reconnaissance proprement dite nous n'avons pas développé excessivement cet aspect. Il a d'ailleurs fait l'objet de nombreuses recherches en ce qui concerne la reconnaissance de caractères pour le texte traditionnel. Il suffirait d'adapter ces recherches aux symboles mathématiques spéciaux: $\sum$, $\int$, etc...

La matrice d'alternatives

La matrice d'alternatives que nous utilisons est une matrice auxilliaire, elle ne prétend pas être exhaustive. Il peut s'avérer utile de compléter celle-

ci. Les transformations que nous pourrions faire à cette matrice devant être le fruit d'utilisations répetées de notre programme et de comparaisons entre les résultats obtenus et l'image de base, c'est au fil du temps que les améliorations y seront apportées.

Il est aussi évident que l'utilisation de notre programme sur des formules manuscrites ou édités dans d'autres traitements de texte permettra la mise au point de cette matrice.

Table des matières

Table des figures

Bibliographie

[1] Libtiff tutorial. www.libtiff.org.

[2] Librairie libtiff. home.earthlink.net/ritter/tiff.

[3] P. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. Dans *Proceedings SPIE Conference on Visual Communications and Image Processing IV*, pages 852–863, Philadelphie, Etats-Unis, novembre 1989.

[4] Alexandra Dessy et Marie-Pierre Fivet. Mémoire pour l'obtention du grade de licencié en sciences mathématiques. *Application des méthodes de classification supervisée à la reconnaissance de caractères dactylographiés dans un texte bruité*, Année académique 1997-1998.

[5] Catherine Harmel et Anne Marchant. Mémoire pour l'obtention du grade de licencié en sciences mathématiques. *Algorithme de reconnaissance de formes non convexes basé sur l'analyse discriminante*, Année académique 1998-1999.

[6] Anil K. Jain et Ju Bin. Page segmentation using document model. Dans IEEE Computer Society, éditeur, *Proceedings of Fourth International Conference on Document Analysis and Recognition*, volume 1, pages 34–38, août 1997.

[7] A. Kacem, A. Belaid, et Ahmed M. EXTRAFOR: Automatic EXTRAction of mathematical FORmulas. Dans *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, pages 107–110, Bangalore, Inde, septembre 1999. IEEE Computer Society Press.

[8] Murat Kunt, Giovanna Coray, Goesta Granlund, Jean Paul Haton, Rolf Ingold, et Michel Kocher. *Traitement de l'information*, volume Reconnaissance des formes et analyse de scènes. Presse polytechniques et universitaires romandes, 2000.

[9] Stéphane Lavirotte. Thèse pour l'obtention du grade de docteur en sciences informatiques. *Reconnaissance structurelle de formules mathématiques typographiées et manuscrites*, 2000.

[10] Hsi-Jian Lee et Jiumn-Shine Wang. Design of a mathematical expression recognition system. Dans *Proceedings of the Third International Conference on Document Analysis and Recognition*, pages 1084–1087, Montréal, Canada, août 1995. IEEE Computer Society Press.

[11] S.U. Lee, S.Y. Chung, et R.H. Park. A comparative performance study of several global tresholding techniques for segmentation. *Computer Vision, Graphics, and Image Processing*, 52(2):171–190, novembre 1990.

[12] Laurence O'Gorman et Rangachar Kasturi. *Executive Briefing: Document Immage Analysis.* IEEE Computer Society, 1997.

[13] M. Okamoto et B. Miao. Recognition of mathematical expressions by using the layout structure of symbols. Dans *Proceeedings of First International Conference on Document Analysis and Recognition*, volume 1, pages 242–250, Saint Malo, France, octobre 1991. IEEE Computer Society.

[14] M. Okamoto et A. Miyazawa. An experimental implementation of a document recognition system for papers containing mathematical expressions. Dans H.S. Baird, H. Bunke, et K. Yamamoto, éditeurs, *Structured Document Image Analysis*, pages 36–53, Heidelberg, Allemagne, 1992. Springer Verlag.

[15] M. Okamoto et H.M. Twaakyondo. Structure analysis and recognition of mathematical expressions. Dans *Proceedings of Third International Conference on Document Analysis and Recognition*, pages 430–437, Montréal, Canada, août 1995.

[16] P.K. Sahoo, S. Soltani, A.K.C Wong, et Y.C. Chen. A survey of thresholding techniques. Dans *Computer Vision Graphics Image Processing*, volume 41, pages 233–260. Academic Press, 1988.

[17] P. Soille. *Morphological image analysis.* Springer Verlag New York, 1999.

[18] W.Y. Wu, M.J.J.J. Wang, et C.M. Liu. Performance evaluation of some noise reduction methods. *Computer Vision, Graphics, and Image Processing. Graphical Models and Image Processing*, 54(2):134–146, mars 1992.